A UML 2.0 Profile for SystemC: Toward High-level SoC Design*

E. Riccobene
Univ. di Milano
Dip. di Tec. dell'Inf.
Bramante 65, Crema, Italy
riccobene@dti.unimi.it

P. Scandurra
Univ. di Catania
Dip. di Mat. e Inf.
A. Doria 6, Catania, Italy
scandurra@dmi.unict.it

A. Rosti and S. Bocchio
STMicroelectronics Lab R&I
C.d.Colleoni 20041 Agrate
Brianza, Italy
{alberto.rosti,sara.bocchio}
@st.com

ABSTRACT

In this paper we present a UML 2.0 *profile* for the SystemC language, which is a consistent set of modeling constructs designed to lift both structural and behavioral features (including events and time features) of the SystemC language to UML level. The main target of this profile is to provide a means for software and hardware engineers to improve the current industrial Systems-on-a-Chip (SoC) design methodology joining the capabilities of UML and SystemC to operate at system-level.

Categories and Subject Descriptors: B.m [Miscellaneous]: Design management

General Terms: Design, Languages.

Keywords: UML, UML profiles, SystemC, embedded systems, model-driven System on Chip design.

1. INTRODUCTION

To specify, design, and implement complex Systems on Chip (SoC) that include hardware and software parts, the Electronic Design Automation (EDA) communities are pushing a shift in design entry level. This additional abstraction step implies that an increasing amount of system design will be specified using system-level languages. Therefore, an issue facing SoC designers is to decide which system-level language to use and how the verification tasks will be accomplished. Any language proposed to support the SoC design must address two important characteristics: the integration of multiple heterogeneous models and the ability to work at high levels of abstraction. Furthermore, a new design process is needed; it should allow modular, component-based approach to both hardware and software design to face the complexity of SoCs.

A *new frontier* system-level languages proposal [4, 8] consists in extending standard lightweight software modeling languages like UML [17] in order to apply them as higher languages operating in synergy with some other lower level languages. The work presented in this paper can be seen as an effort further this new direction.

We defined a UML 2.0 profile for SystemC [16], able to capture both the structural and the behavioral features of the language. This UML profile defines a language that allows to specify, analyze, design, construct, visualize and document the software and hardware artifacts in a SoC design flow. Leveraging the joint capabilities of UML and SystemC we provide a modeling framework for systems in which high-level functional models can be refined down to an implementation language. The choice of SystemC as implementation language is intentional, and mainly motivated by the fact that SystemC is becoming one of the most important players in the future of SoC design and verification. In [4], we discuss how the current SoC design flow at STMicroelectronics can be improved by UML and SystemC.

The paper is organized as follows. In section 2, we introduce the SystemC language, while in section 3 we sketch how to define a UML profile for a given language. In section 4 we present the UML 2.0 profile for SystemC. Finally, in section 5 we provide an overview of the case tool support for our profile, discuss about systems we are modeling as case studies, and cite some related work.

2. SystemC OVERVIEW

SystemC [13, 16] is an open standard controlled by the major companies in the EDA (Electronic Design Automation) industry. It is a promising system-level design language intended to support the description and validation of complex systems in an environment completely based on the C++ programming language.

The SystemC language is defined in terms of a layered-architecture. Built on top of the C++ language, the *Core Language* and *Data Types* are the so called *core layer* (or layer 0) of the standard SystemC. The *Primitive Channels* represents, instead, the layer 1 of SystemC; it comes with a predefined set of interfaces, ports and channels.

A design of a system is essentially broken down into a containment hierarchy of *modules*. A *module* is a container class able to encapsulate *structure* and *functionality* of hard-

*This work has been supported in part by the project *Tecniche e metodologie di progetto, documentazione, verifica e validazione per i sistemi di IP (Intellectual Property)* at STMicroelectronics.

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

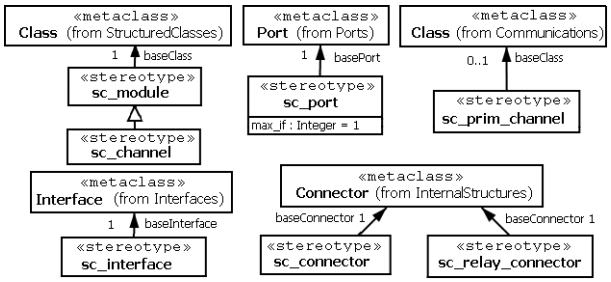

Figure 1: Structure Stereotypes Definition

ware/software blocks. Each module may contain *variables* as simple data members, *ports* for communication with the surrounding environment and *processes* for performing module's functionality and expressing concurrency in the system. Two kind of processes are available: *method* processes and *thread* processes. They run concurrently in the design and may be sensitive to *events* which are notified by other processes. A port of a module is a proxy object through which the process accesses to a channel interface. The *interface* defines the set of access functions for a channel, while the channel provides the implementation of these functions to serve as a container to encapsulate the *communication* of blocks. There are two kind of channels: *primitive* channels and *hierarchical* channels. Primitive channels do not exhibit any visible structure, do not contain processes, and cannot (directly) access other primitive channels. A hierarchical channel is a module, i.e., it can have structure, it can contain processes, and it can directly access other channels.

3. UML PROFILES

Recently, UML [17] has been used in areas for which it was not originally intended. Many proposals have been arisen for extending UML for specific domains (as banking, telecommunications, aerospace, real-time systems, etc). In UML, these extensions are called *profiles*. A UML profile is a set of *stereotypes*. Each stereotype defines how the syntax and the semantics of an existing UML construct (a class of the UML metamodel) is extended for a specific domain terminology or purpose. A stereotype can define additional semantic *constraints* – the *well-formedness rules* – expressed as OCL (Object Constraint Language) [12] formula over its base metaclass, as well as *tags* to state additional properties.

For defining profiles, UML 2.0 is endowed with a standard graphical notation. A package with keyword $\ll profile \gg$ denotes a profile. Within the profile package, a class of the UML metamodel that is extended by a stereotype is represented as a conventional class with the optional keyword $\ll metaclass \gg$. A stereotype is depicted as a class with the keyword $\ll stereotype \gg$. The extension relationship between a stereotype and a metaclass is depicted by an arrow with a solid black triangle pointing toward the metaclass. In Figures 1 and 5 several examples of stereotypes definition are reported.

At model level, when a stereotype is applied to a model element (an instance of a UML metaclass), an instance of a stereotype is linked to the instance of the corresponding UML metaclass. From a notational point of view, the name of the stereotype is shown within a pair of guillemets above or before the name of the model element and the eventual

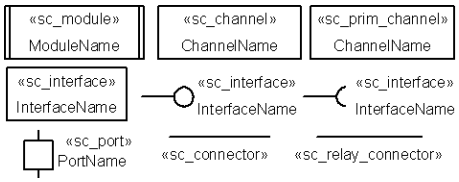

Figure 2: Structure Stereotypes Notation

tagged values displayed inside or close as name-value pairs. Fig. 2 reports some examples of stereotypes application.

4. A UML 2.0 PROFILE FOR SystemC

This section introduces our UML profile for the SystemC language based on the UML 2.0 specification [17] and on the SystemC 2.0 specification [16].

The profile is defined at two distinct levels – the *SystemC core layer* (or layer 0) and the *SystemC layer of predefined channels, ports and interfaces* (or layer 1) – which reflect the layered-architecture of SystemC. The core layer is the foundation – *the basic SystemC profile* – upon which specific libraries of model elements or also other modeling constructs can be defined. It is logically structured as follows:

– The **Structure and Communication** part defines stereotypes for the primitive building blocks of SystemC. They are used in various UML structural diagrams (such as UML class diagrams and composite structure diagrams) to represent hierarchical structures and communication blocks made of modules, interfaces, ports and channels.

– The **Behavior and Synchronization** part defines stereotypes which lead to a variation of the UML method state machines, the *SystemC Process State Machine* (see [2]), to allow high level specification of the behavior of SystemC processes (methods and threads) within modules and channels.

– The **Data types** part defines UML classes for representing the set of SystemC data types. Since in UML a data type is a special kind of *classifier* in our profile the SystemC data types are represented in terms of UML classes, all contained in a stand alone package imported by the SystemC profile's package.

The SystemC layer of predefined channels, ports and interfaces provides concepts for the layer 1 of SystemC. These concepts are implemented both as a *class library*, modelled with the basic group of stereotypes of the SystemC core layer, and as a group of stand alone stereotypes – *the extended SystemC profile* – which specializes the basic profile.

In the following sections, we describe the most significant stereotypes of the SystemC core layer giving the semantics in an informal way. We intentionally leave out the OCL constraints due to lack of space. The complete UML profile definition for SystemC is in [3]. Moreover, we assume the reader is familiar with UML 2.0 and SystemC.

4.1 Structure and Communication

Fig. 1 shows the stereotypes definition using the standard notation presented in sect. 3, while Fig. 2 depicts their notation when they are applied at model level.

A SystemC *module* is modelled as an extension of an *active structured class* of the UML with `sc_module` stereotype; a composite structure diagram can be further associated to the module class to represent its internal structure (if any).

A SystemC *interface* is mapped to a UML interface with

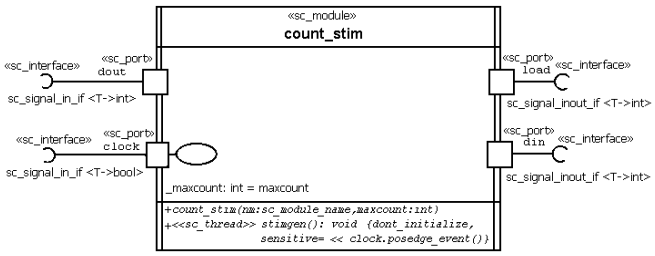

Figure 3: Example of a module, its attributes, functions, processes and ports

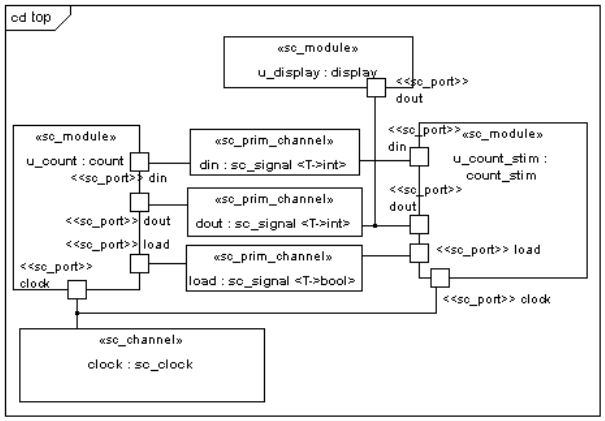

Figure 4: Example of a Composite Structure

`sc_interface` stereotype, and uses its (longhand/shorthand) notation.

The `sc_port` stereotype maps the notion of SystemC port directly to the notion of UML port, plus some constraints to capture the concepts of *simple*, *multi* and *behavior* port. The tag `max_if` defined for the `sc_port` stereotype specifies the maximum number of channel interfaces that may be attached to the port. The type of a port, namely its required interface, is shown with the socket icon attached to the port (see Fig. 2 and Fig. 3).

Since UML ports are linked by connectors, a SystemC *connector* (binding of ports to channels) is provided as extension of the UML connector, by the `sc_connector` stereotype. A relay connector – the `sc_relay_connector` stereotype – is another construct defined to represent the port-to-port binding between a parent port and a child port[1].

The `sc_prim_channel` and `sc_channel` stereotypes define, respectively, a SystemC primitive and hierarchical channel as extension of a simple UML class that implements a certain number of interfaces (i.e. the provided interfaces of the channel). A hierarchical channel can further have *structure* (including ports), it can directly access other channels, and it can contain processes.

Fig. 3 shows a module class `count_stim` containing a thread process `stimgen`, two input ports `dout` and `clock`, and two output ports `load` and `din`. In particular, `clock` is a behavior port since it provides events that trigger the `stimgen` thread process within the module.

Instead, the composite structure diagram in Fig. 4 represents the internal structure of a module `top`. This module is used to assemble parts (groups of instances) of the inner modules `count`, `count_stim` and `display`, and of the two `clock` and `sc_signal` channels – predefined SystemC channels –. Note that in this example, all connectors are of kind `sc_connector`, even if the stereotype keyword is not shown to make the diagram more readable.

[1]A port-to-port connection is the binding of a module port (parent port) to a lower level module port (child port).

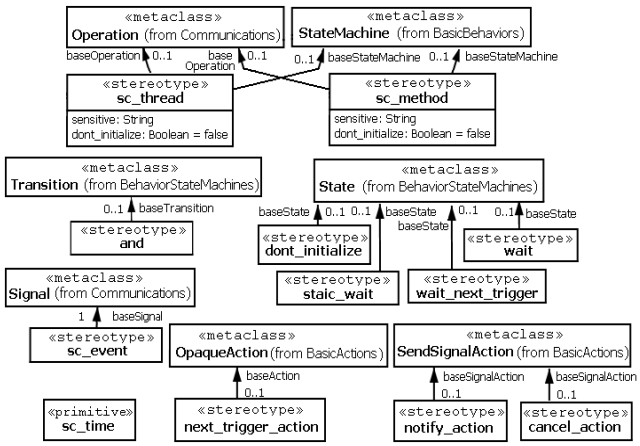

Figure 5: Behavior Stereotypes Definition

4.2 Behavior and Synchronization

Fig. 5 shows the stereotypes definition for the SystemC constructs used to model the behavioral aspects of a system.

Two kinds of processes are available: `sc_method` and `sc_thread`. Both behave like an operation with no arguments and no return type. Each kind of process has a slight different reactive behavior. We defined [2] a new graphical formalism, called *SystemC Process State Machine*, exploiting the UML 2.0 *method* state machine [17], to allow high level specification of the functionality of the reactive SystemC processes and generation of efficient and compact executable SystemC code.

Our profile provides stereotype definitions (`wait`, `static_wait`, `and`, `wait_next_trigger`, `next_trigger_action`, `dont_initialize`) to model the *static* and *dynamic sensitivity* mechanism of a process behavior.

The `sc_event` stereotype models a SystemC *event* in terms of a UML signal (instance of the class `SignalEvent`).

The `notify_action` stereotype can be applied to an UML action to model the SystemC function `notify` used to notify events. The `cancel_action` stereotype can be applied to an UML action to model the SystemC function `cancel` used to eliminate a pending notification of an event.

The `sc_time` type is used to specify concrete time values used for setting clocks objects, or for the specification of UML time triggers.

As part of the UML profile for SystemC, the control structures `while`, `if-then-else`, etc. are represented in terms of special stereotyped junction or choice pseudostates.

In addition to the stereotypes presented above, we defined particular *abstract behavior patterns* of state machines to specify the behavior of SystemC processes [2].

The left side of Fig. 6 depicts one of these behavior patterns. It corresponds to a thread process which: (i) has both a static (the event list $e_{1s}, \ldots, e_{Ns}$) and a dynamic sensitivity (represented by the state `WAITING FOR e*` with

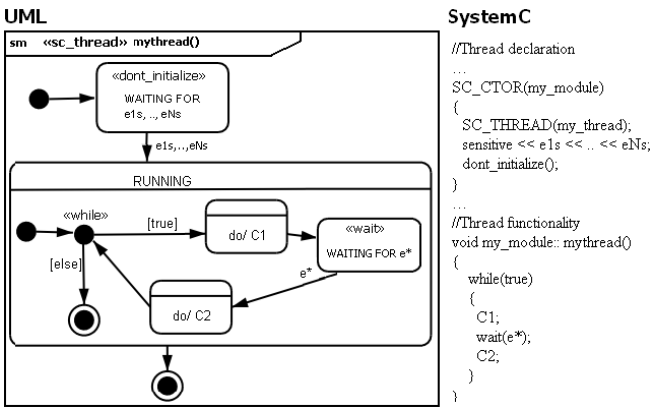

Figure 6: A Thread Process pattern

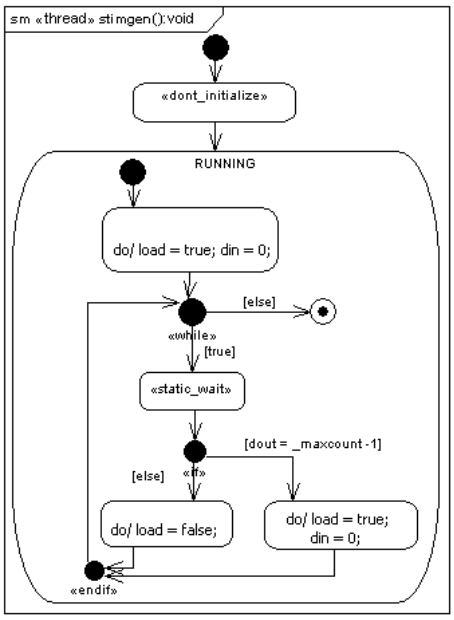

Figure 7: A thread process state machine

the stereotype `wait`)[2], (ii) runs continuously (its functionality is enclosed by an infinite `while` loop), and (iii) is not initialized (the state with the `dont_initialize` stereotype follows the top initial state). The right side of Fig. 6 shows the corresponding SystemC pseudo-code.

Fig. 7) show the state machine of the `stimgen` thread process of the module example given in Fig. 3, as a realist example of the behavior pattern presented in Fig. 6.

5. CONCLUSIONS AND RELATED WORK

To support our design methodology a necessary condition is the provision of a UML 2.0 visual modeling tool empowered by our UML SystemC profile. Our current implementation is based on the Enterprise Architect (EA) tool [5], however any other tool supporting UML 2.0 and UML profiles could be appropriate. The design of a system can easily evolve from an initial functional UML model (corresponding to a functional C/C++ specification in the traditional design methodology), adding new stereotyped elements and diagrams, and introducing packages for modularizing and decomposing issues. Moreover, as the EA tool allows reverse-engineering of C/C++ code, the design activity can also start from the UML model obtained by reverse engineering from an existing C/C++ functional specification, and then evolves applying stereotypes and new stereotyped elements as well. In addition, we endowed the EA tool with an add-in for generating SystemC code – *forward engineering* – from UML models, for both structural and behavioral aspects. We have already developed several small case studies to test the expressive power of our language in representing a variety of architectural and behavioral aspects. Currently, to evaluate the efficacy of our language to cope with complex architectural designs, we are modeling the On Chip Communication Network (OCCN) [11] library.

The possibility to use UML 1.x for SoC design [6, 7] started at Cadence Design Systems in 1999. [1] attempts to define a UML profile for SystemC based on UML 1.4; but, as other proposals, no code generation capabilities for behavioral information are considered. [9] faced the problem of code generation but does not rely on a UML profile definition. Fujitsu [14] developed a new SoC design methodology that employs UML and C/C++/SystemC programming languages. The SysML (Systems Modeling Language) Partners [15] are collaborating to customize UML 2.0 to define a modeling language for systems engineering applications. Since SysML preserves the basic semantics of UML 2.0 diagrams (state formalism, e.g., are unchanged), our UML profile can be thought (and effectively made) a customization of SysML rather than UML.

[2]Note that the notation used for the `wait` state in the state machine pattern given in Fig. 6 stands for a shortcut to represent a generic `wait(e*)` call where the event `e*` matches several cases. See [2] for more details.

6. REFERENCES

[1] F. Bruschi and D. Sciuto. SystemC based Design Flow starting from UML Model. In *Proc. of ESCUG'02*.

[2] E. Riccobene, P. Scandurra. Modelling SystemC Process Behaviour by the UML Method State Machines. In *Proc. of RISE'04*. LNCS 3475, Springer.

[3] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio. A UML 2.0 Profile for SystemC. ST Microelectronics Technical Report AST-AGR-2005-3.

[4] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio. A SoC Design Methodology Based on a UML 2.0 Profile for SystemC. In *Proc. of DATE'05*. IEEE Computer Society Press.

[5] The Enterprise Architect tool: www.sparxsystems.com.au/.

[6] G.Martin. UML and VCC. White paper, Cadence Design Systems, Inc, Dec. 1999.

[7] G.Martin, L.Lavagno, J.L.Guerin. Embedded UML: a merger of real-time UML and co-design. In *Proc. of CODES'01*.

[8] A. Habibi and S. Tahar. A Survey on System On a Chip Design Languages. In *Proc. of IWSOC'03*. IEEE Computer Society Press.

[9] K. D. Nguyen, Z. Sun, P. S. Thiagarajan and Weng-Fai Wong. Model-driven SoC Design Via Executable UML to SystemC. In *Proc. of RTSS'04*. IEEE Computer Society Press.

[10] OMG. The Model Driven Architecture (MDA). http://www.omg.org/mda/.

[11] OCCN Project Web site: http://occn.sourceforge.net/.

[12] OMG. UML 2.0 OCL Specification, ptc/03-10-14.

[13] The Open SystemC Initiative: www.systemc.org.

[14] Q.Zhu, R.Oishi, T.Hasegawa, T.Nakata. System on chip validation using UML and CWL. In *Proc. of CODES*, 2004.

[15] SysML Partners web site: http://www.sysml.org/.

[16] T. Grotker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*. Kluwer Publisher, 2002.

[17] OMG. UML 2.0 Superstructure, ptc/04-10-02.